Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Simulace GPS signálu

Plzeň, 2013 Jan Masopust

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 1.5.2013 …....….....

Abstract

GPS signal simulation

This bachelor thesis deals with creating of system that allows to move around the world and generate the GPS coordinates specifying position. In first part describes possibilities of moving around the Earth, simulating of movement and vizualization. In second part is a description of the implementation and testing of the system.

Obsah

1. Úvod

Ve většině mobilních zařízení, jako jsou mobilní telefony, laptopy nebo tablety se používají aplikace, které jsou přímo ovlivňovány pohybem zařízení ve světě. Při vývoji takovýchto aplikací není možné volně pohybovat zařízením na velké vzdálenosti, k tomu se dají použít systémy na simulaci pohybu.

Úkolem této bakalářské práce je právě takovýto systém vyvinout. Tedy navrhnout a implementovat Java program, který bude simulovat pohyb objektu po světě a generovat zeměpisné souřadnice určující jeho polohu. Je tedy nutné seznámit se s fungováním navigačního systému a najít možnosti na zadávání cest, po kterých se bude objekt pohybovat. Poté bude nutné přejít k samotnému pohybu, zjistit jakými způsoby lze pohyb po Zemi simulovat a nakonec je zapotřebí najít způsob, jak získané výsledky vizualizovat, aby bylo možné kontrolovat správné fungování.

Práce je rozdělena na dvě části, přičemž v první části jsou uvedeny možnosti, jakými bylo možné jednotlivé části implementovat, a teoretické podklady nezbytné k úspěšnému vytvoření. V druhé části jsou již vybrány vhodné nástroje, popsán důvod a způsob jejich použití a popsána implementace samotného programu a jeho testování.

Teoretická část

2. Technologie GPS

2.1 Co je GPS?

GPS neboli Global Positioning system neboli globální polohový systém je družicoví navigační systém, který je vlastněn a provozován ministerstvem obrany USA. Pomocí tohoto systému je možné určit přesnou polohu a čas na Zemi. Díky rovnoměrnému rozmístění družic na oběžné dráze je systém přístupný kdekoliv na planetě a funguje 24 hodin denně.[1]

2.2 Historie GPS

Doba vzniku prvních satelitních navigačních systémů se datuje od druhé poloniny 20. století. Důvodem, proč začali vznikat takovéto systémy, byly armádní zájmy, konkrétně navigace vojenských plavidel na moři. V roce 1960 začalo americké námořnictvo umisťovat na oběžnou dráhu družice pro určování polohy, systém se tehdy jmenoval TRANSIT.

Jeho nejúspěšnějším nástupcem se stal právě systém GPS. Počátky jeho vývoje spadají do roku 1973, kdy se spojili dvě jiné metody na určování polohy Timation a 621B. Kde Timation sloužil k přesnému určování času a 621B k určování polohy. Do roku 1978 byly k dispozici na oběžné dráze již čtyři družice a bylo tedy možné začít s testováním fungování. Polohu bylo možné určovat jen v malé části Ameriky a jen po omezenou dobu. A do roku 1979 bylo vypuštěno celkem 11 družic.

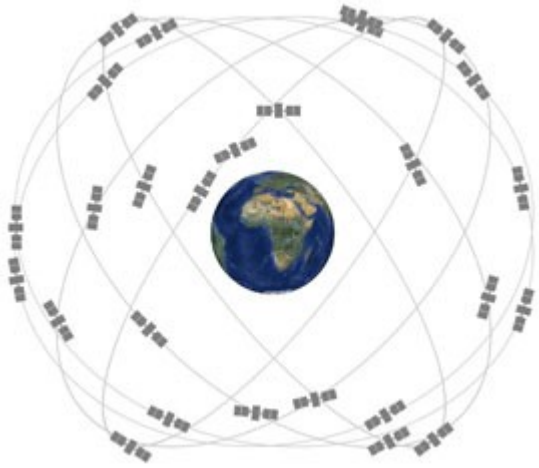
V letech 1979 – 1985 začala výstavba řídících a monitorovacích středisek potřebných k řízení systému a také byl zahájen vývoj přijímačů. V roce 1989 se začali na oběžnou dráhu vypouštět další družice, aby bylo možné systém používat na celém světe a kdykoliv. Do roku 1998 bylo na oběžnou dráhu vypuštěno 28 družic, které byli schopné pracovat bez kontaktu s řídícími středisky až 180 dní, ale již v roce 1994 byl celý systém GPS s 24 družicemi na oběžné dráze prohlášen za plně funkční.[6]

2.3 Struktura GPS

Systém GPS je tvořen třemi segmenty: kosmickým, řídícím a uživatelským.

Kosmický segment se skládá z družic vysílajících radiové signály pro uživatele. Je řízený letectvem spojených států, které zaručuje, aby bylo přístupných alespoň 24 satelitů po 95% času. V této chvíli je na oběžné dráze 31 funkčních satelitů a 3-4 satelity vyřazené, které je možné v případě potřeby znovu aktivovat a používat.

Satelity létají na střední oběžné dráze ve výšce zhruba 20350 kilometrů nad mořem a každý satelit oběhne Zemi dvakrát za den. Družice létají po šesti rovnoměrně rozložených oběžných drahách kolem Země, z nichž na každé jsou minimálně čtyři družice. Takovéto rozložení zajistí, že jsou na jakémkoliv místě na světě vidět nad obzorem alespoň čtyři družice, což umožňuje trojrozměrné určení polohy.



Obrázek 2.1 – Rozmístění družic kolem Země

Řídící segment se skládá z celosvětové sítě pozemních zařízení, které sledují GPS satelity, monitorují jejich vysílání, provádí analýzy a posílají jim příkazy a data. Aktuálně obsahuje řídící segment hlavní řídící středisko, záložní hlavní středisko, 12 řídících vysílačů a 16 monitorovacích míst. Monitorovací stanice jsou rozmístěné po obvodu Země blízko rovníku.

Hlavní řídící stanice se nachází v Coloradu a plní základní funkce řídícího segmentu, vytváří a nahrává navigační zprávy a zajišťuje funkčnost a přesnost systému. Přijímá navigační zprávy od monitorovacích stanic, pomocí kterých vypočítává přesnou polohu

satelitů na oběžné dráze a poté nahrává tyto data zpátky na satelity přes řídící stanice.

Uživatelský segment se skládá z pasivních zařízení, které vlastní jednotlivý uživatelé. To, že jsou pasivní znamená, že nekomunikují s žádnou družicí, pouze přijímají signál. Pomocí tohoto signálu určují svou polohu a čas. Díky tomu, že jsou tato zařízení pasivní a nemusí tedy komunikovat s družicemi, může tento systém používat prakticky neomezené množství uživatelů. [1]

2.4 Fungování GPS

Družice vysílá rádiový signál, který obsahuje informace o její poloze a stavu a také přesný čas, který získávají z atomových hodin, které sebou nesou. Tento signál cestuje prostorem rychlostí světla až k jednotlivým zařízením. Zařízení přijme signál a zapíše si čas jeho příchodu.

Následně spočte vzdálenost od družice. Ta se spočte tak, že se vynásobí rychlost světla a doba cesty, kde doba cesty je rozdíl mezi časem vyslaným družící a časem přijetí signálu. Jakmile má zařízení takto spočtené vzdálenosti alespoň od 4 různých družic, může určit svou polohu na Zemi. [7]

2.5 Použití GPS

Námořní doprava

GPS je pro námořníky neocenitelný prostředek, protože umožňuje snadnou navigaci a určení polohy, což je na otevřeném moři velmi obtížné. Umožňuje také snadné směrováni provozu, to šetří čas a palivo. Zároveň také pomáhá při koordinaci záchranných a pátracích akcích.

Zemědělství

Kombinací GPS a geografických informačních systémů(GIS) se zemědělství stává mnohem produktivnější. Aplikace využívající tyto technologie jsou schopné efektivně plánovat a mapovat terén, to umožňuje efektivní sázení, chemický postřik nebo hnojení a to vše i za zhoršených podmínek jako je déšť, mlha nebo tma.

Letectví

Piloti po celém světě využívají GPS ke zvýšení efektivity a bezpečnosti letů. Díky tomu, že GPS systém umožňuje trojrozměrné určování polohy je možné určovat i výšku letounu, to se také využívá při startu, při letu a hlavně při přistávání, kdy lze takto snadno zjistit vzdálenost letadla od přistávací plochy. To je velmi užitečné při zhoršených meteorologických podmínkách.

Životní prostředí

S pomocí GPS a GIS je možné také sledovat vývoj životního prostředí. Je díky tomu možné shromažďovat data o vývoji prostředí v nějaké oblasti a regulovat tam například těžbu. Zároveň je možné systém využít i při různých přírodních katastrofách. Například při velkých požárech je tak možné sledovat a určovat jejich postup a podle toho koordinovat záchranné složky.

Železniční doprava

GPS se používá i k určení přesné polohy vlaků a dalších zařízení na tratích. V kombinaci s dalšími senzory na drahách dokáže odvrátit kolize a tím zvyšovat bezpečnost. Dále také umožňuje zajistit plynulost dopravy a tím minimalizovat zpoždění. Je tak možné vytvářet automatické systémy, které jsou schopné správně řídit pohyb na železnici.

Rekreace

GPS se rozšířila i mezi obyčejné lidi, kteří ji využívají při různých outdoorových aktivitách. Dnes je GPS přístupný i v mobilních zařízeních, takže si ho může dovolit používat téměř každý. Turistům slouží k určení přesné polohy míst, která chtějí navštívit. Díky schopnosti ukládat si procházenou trasu je možné stáhnout si trasu do počítače a sdílet ji s dalšími lidmi. GPS se stala součástí i různých her jako je například geocaching, což je sport, při kterém se hledá „poklad“ podle zadaných GPS souřadnic.

Silniční doprava

Díky přesnosti a dostupnosti GPS zařízení je možné zvýšit efektivnost a bezpečnost i na dálnicích a silnicích. Spolu s dalšími systémy je možné spočítat optimální cestu mezi dvěma body a to i s ohledem na zácpy a podobné překážky, bránící plynulé dopravě. GPS se nepoužívá pouze při dopravě zboží, ale také již v osobních automobilech. Je

možné dokonce kontrolovat stav silnic a koordinovat tak jejich výstavbu a nebo opravy.

Vesmír

GPS usnadňuje určování polohy ostatních družic a jejich formace na oběžných drahách. Dále také pomáhá při různých vědecký projektech prováděných ve vesmíru.

Geodézie a mapování

Tato oblast byla mezi prvními, kteří začali GPS využívat a to hlavně k zpřesnění údajů na mapách. Na rozdíl od klasických technik měření, není GPS svázáno viditelností na zkoumaném území. Je možno přesně mapovat a modelovat okolní fyzický svět.

Čas

Kromě poskytování polohy umí GPS systém určovat i přesný čas a to díky tomu, že jsou na každém satelitu umístěny atomové hodiny. Uživatelé tak můžou určovat čas s přesností do nanosekund, bez toho aniž by vlastnili atomové hodiny. Přesný čas je důležitý v různých hospodářských odvětvích jako jsou komunikační systémy, rozvod elektrické energie nebo bankovnictví. Umožňuje dokonce efektivnější využívání rádiového spektra telefonních sítí.[1]

2.6 GPX

GPX nebol GPS eXchnage Format je formát souborů k ukládání bodů a tras na Zemi a je založen na formátu XML. Lze ho snadno přenášet mezi různými GPS zařízením nebo lze přenést do počítače. V počítači je možné si trasu prohlédnout a editovat a případně sdílet mezi dalšími lidmi. Je možné si takto vytvořit na počítači úplně novou cestu, kterou lze potom nahrát do GPS zařízení.

Existují dvě verze GPX:

od roku 2002 je to verze 1.0

od roku 2004 je to verze 1.1

V verzi 1.1 jsou na rozdíl od předchozí verze všechny obecné informace v elementu *metadata*, elementy *url* a *urlname* nahradil *link* a rozšiřující elementy jsou v *extension*.

Stejně jako jakékoliv jiné XML, je možné načítat GPX v Jave různými parsery:

2.6.1 DOM

DOM je standard pro přístup a manipulaci s XML dokumenty. Převádí XML na stromovou strukturu, kterou lze snadno procházet a měnit jí. Tento způsob je však nevhodný pro velké soubory, protože DOM načítá celý strom do paměti a to je v případně velkého dokumentu problém , protože se do paměti nemusí vejít.

2.6.2 SAX

SAX je alternativním způsobem načítání XML souborů k DOMu. Největší výhodou je, že SAX nenačítá do paměti celý strom, ale při procházení souboru zasílá zprávy, které je možné odchytávat. SAX je obecně mnohem rychlejší než DOM, ale na rozdíl od něj se neumí vracet k již přečteným elementů nebo je editovat.

2.6.3 JAXB

JAXB je nástroj, který umožňuje vývojářům vytvořit knihovnu na základě struktury XML souboru. Po načtení knihovny do Java projektu lze s XML pracovat jako s obyčejnými objekty a při používání pokročilých editorů nemůže programátor téměř udělat chybu, protože správné procházení strukturou XML kontroluje překladač.

3. Metody počítačové simulace

V případě, že chceme zjistit chování nějakého systému v různých situacích a není možné to reálně vyzkoušet nebo by to bylo příliš nákladné, používá se právě simulace. Při simulaci se pracuje s matematickým modelem testovaného systému, který má parametry potřebné k simulaci. Není potřeba znát a nastavovat všechny vlastnosti, protože některé simulaci vůbec neovlivňují. Například při simulaci pohybu auta nebude zapotřebí zadávat jeho barvu.

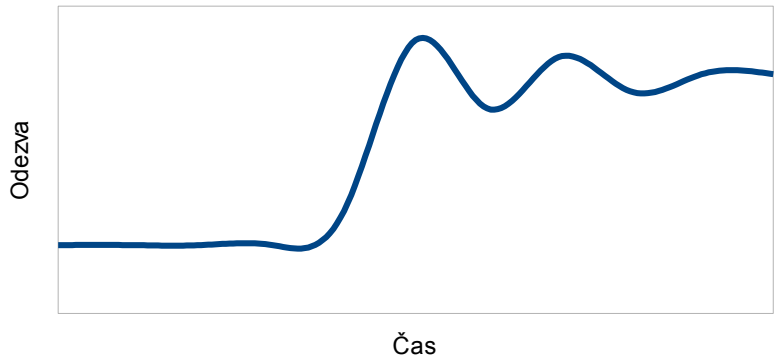
Různé situace v modelovaném systému se získávají změnou různých vstupních parametrů, jako například při zmíněném pohybu auta můžeme získat různé situace tím, že změníme jeho rychlost. Takto získané výsledky se většinou analyzují a vyvozují se z nich závěry, které je poté možné znovu aplikovat na reálný svět.

Simulace je možné rozdělit do dvou skupin, na simulaci diskrétní a simulaci spojitou. Tyto simulace se liší způsobem modelování času. [4]

3.1 Spojitá simulace

Spojitá simulace se vyznačuje tím, že nepřetržitě sleduje odezvu nějakého systému v průběhu času podle soustavy rovnic, které typicky zahrnují diferenciální rovnice. Takto lze zjistit stav modelu v jakékoliv okamžiku simulace.

Jako příklad lze uvést: Odezvu kola automobilu na jednotkový skok.

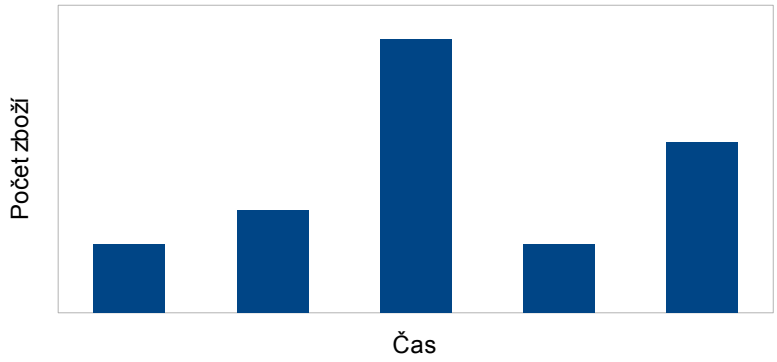


Graf 3.1 – Příklad výstupu spojité simulace

3.2 Diskrétní simulace

Diskrétní simulace se od spojité liší především v tom, že se její stav mění s krokem. Krok může být pravidelný, kde se model přepočítá typicky po časových intervalech, nebo nepravidelný, kde se parametry modelu přepočítávají, když nastane nějaká událost. To co se děje mezi jednotlivými kroky je pro simulaci nezajímavé.

Jako příklad lze uvést: Počet zboží přijatého na sklad v průběhu dne. Následující krok nastane, když přijede další kamión.



Graf 3.2 – Příklad výstupu diskrétní simulace

4. Pohyb po povrchu Země

Následující vzorce slouží k výpočtu hodnot souvisejících pohybem po Zemi. Rovnice však počítají s kulovým tvarem Země to znamená, že ignorují fakt, že Země je elipsoid, to může způsobovat chyby až do 0,3%.

V případě že je zadán výchozí bod, směr neboli azimut a vzdálenost a je zapotřebí získat výsledný bod, používají se tyto rovnice:

$$\phi_2 = asin(\sin(\phi_1)\cdot\cos(d/r) + \cos(\phi)\cdot\sin(d/R)\cdot\cos(\theta))$$

$$\lambda_2 = \lambda_1 + atan2(\sin(\theta)\cdot\sin(d/R)\cdot\cos(\phi_1), \cos(d/R) - \sin(\phi_1)\cdot\sin(\phi_2))$$

Vzorec 4.1

Kde ϕ_1 je zeměpisná šířka a λ_1 zeměpisná délka výchozího bodu. θ je azimut, *d* je vzdálenost a *R* je poloměr země. Z toho získáme zeměpisnou šířku ϕ_2 a zeměpisnou délku λ_2 koncového bodu.

Pokud je zadán výchozí a koncový bod a je potřeba spočítat azimut mezi nimi, použijí se následující vzorec:

$$\theta = atan2(\sin(\Delta\lambda)\cdot\cos(\phi_2), \cos(\phi_1)\cdot\sin(\phi_2) - \sin(\phi_1)\cdot\cos(\phi_2)\cdot\cos(\Delta\lambda))$$

Vzorec 4.2

Kde $\Delta\lambda$ je rozdíl zeměpisných délek, ϕ_1 je zeměpisná šířka a λ_1 zeměpisná délka výchozího bodu, ϕ_2 je zeměpisná šířka a λ_2 zeměpisné délka koncového bodu. A výsledkem je požadovaný azimut θ .

Pro výpočet vzdálenosti dvou bodů na povrchu Země se používá vzorec:

$$a = \sin^2(\Delta\phi/2) + \cos(\phi_1)\cdot\cos(\phi_2)\cdot\sin^2(\Delta\phi/2)$$

$$c = 2\cdot atan2\left(\sqrt{(a)}, \sqrt{(1-a)}\right)$$

$$d = R * c$$

Vzorec 4.3

Kde $\Delta\phi$ je rozdíl zeměpisných šířek, ϕ_1 je zeměpisná šířka výchozího bodu, ϕ_2 je zeměpisná šířka koncového bodu a *R* je poloměr Země. A kombinací těchto vzorců se získá vzdálenost *d* mezi dvěma body. [8]

5. OpenStreetMap

OpenStreetMap je projekt shromažďující geografická data celého světa. Tato data jsou nabízená za podmínek Open Data Commons Open Database License a při jejich dodržování jsou všechna komukoliv k dispozici a nejsou na rozdíl od ostatních projektů, které jsou také volně přístupné, jako například Google Map Maker, omezeny žádnými technickými bariérami.

Jakýkoliv uživatel se může zdarma zaregistrovat na stránkách OpenStreetMap a může je editovat, vylepšovat nebo opravovat chyby. Je možné dokonce pomocí svých GPS zařízení mapovat cesty a místa a tyto data nahrávat OpenStreetMap servery. Tento projekt je podporován neziskovou organizací OpenStreetMap Foundation, která zajišťuje, aby se projekt dále rozrůstal a vyvíjel.

OpenStreetMap nejsou pouze volně šiřitelná data, ale také volně šiřitelné open-source programy, které jsou vyvíjeny dobrovolníky po celém světě. Počet lidí, kteří OpenStreetMap používají a podporují stále narůstá a s tím se zvyšuje i kvalita a přesnost map.

5.1 Formát OpenStreetMap

Formát uložení geografických dat vychází z technologie XML. Data se ukládají do jednotlivých prvků:

Node neboli uzel definuje jeden geografický bod pomocí zeměpisné šířky a délky. Mezi volitelné složky, které mohou být k uzlu přidány, patří například nadmořská výška nebo vrstva na mapě. Standardně se používají k definování samostatných objektů ve světě nebo jako části cest.

Jako části cest se používají například na místech, kde se cesty kříží a na daném místě je křižovatka a lze tedy z jedné cesty odbočit na jinou. Naopak se nedávají tam, kde se cesty sice kříží, ale ne jako křižovatka, například třeba jako most přes dálnici.

Way neboli cesta je seřazený seznam uzlů, které reprezentují reálné cesty, silnice a jiné útvary. Cesta může obsahovat 2 až 2000 uzlů a lze je dělit na cesty otevřené nebo uzavřené. Otevřené cesty jsou takové, jejichž počáteční uzel není stejný jako koncový,

což je naprostá většina reálných silnic. Uzavřené cesty zase mají počáteční a koncový uzel stejný, to jsou například kruhové objezdy nebo stěny ohraničující nějakou oblast. Uzavřené cesty lze někdy také interpretovat jako oblasti, tam se řadí například parky nebo budovy.

Relation neboli vztah se skládá z jedné nebo více značek nebo seznamů uzlů a cest a reprezentují nějaký logický celek. Příkladem může být kampus univerzity, ta se skládá z několika budov reprezentovaných oblastmi a z několika cest. Tyto skupiny by neměli být nějak obsáhlé, například seznam všech cest České Republiky by byl příliš obsáhlý. Nedoporučuje se používání více jak 300 položek.

Tag neboli značka se skládá z klíče a hodnoty a slouží k popisování prvků mapy, jako jsou uzly, cesty a vztahy. Značka není element, ale pouze atribut, malá položka s daty, která popisují nadřazený element. Například pokud máme nějakou cestu, ta má určitě omezenou rychlost a ta lze uložit právě pomocí značky: *maxspeed=50*.

5.2 Možnost zobrazování map v Javě

5.2.1 JMapViewer

JMapViewer je Java komponenta pod licencí GPL, která umožňuje snadno zobrazovat OpenStreetMap mapy v Java aplikacích. Komponenta dokáže přibližovat a oddalovat mapu a umí na mapě zobrazovat libovolné značky. Funguje tak, že nahrává ze OpenStreetMap serverů vytvořené bitmapové dlaždice tvořící mapu. Existuje několik způsobů vykreslováni mapy, jako je například Mapnik, a tyto způsoby je možné libovolně měnit.

Vývoj začal v roce 2008 a na jeho vývoji se podílelo několik lidí, ale jeho hlavním strůjcem je Petr Stotz a je stéle podporován. [2]

5.2.2 JXMapViewer

JXMapViewer je open-source swing komponenta pod licencí LGPL a byla vyvinuta skupinou SwingLabs, ale již není nadále podporována. Je to vlastně speciální JPanel s klouzající mapou. Při vytváření GUI je možné ji přidat jako obyčejný panel. Mapy také stahuje jako bitmapové dlaždice z OpenStreetMap serverů.

JXMapViewer je velice podobný jako JMapViewer, ale je o něco komplexnější.

5.2.3 JXMapViewer2

JXMapViewer2 vychází z JXMapViewer a je to jeho nástupce. Opravuje některé chyby a vylepšuje některé funkce, jako je například přibližování, a hlavně je optimalizovaný pro Javu 6. Projekt je pod licencí LGPL a jeho hlavní vývojář je Martin Steiger. [5]

5.2.4 OSM4J

Na rozdíl od ostatních knihoven na zobrazováni mapy je OSM4J mnohem komplexnější a obsahuje mnohem větší soubor metod pro práci s OpenStreetMap. Z mnoha metod je třeba vyzdvihnout například vyhledávání v mapě nebo různé geometrické operace. Největší rozdílem od předchozích knihoven je však možnost pracovat s mapami uloženými na disku počítače a není tedy zapotřebí, ke správnému zobrazování map, neustálé připojení k internetu.

Hlavním vývojářem je Sebastian Kürten, projekt byl založen v roce 2011 a stále je podporován a je pod licencí LGPL. [9]

6. Návrh aplikace

6.1 Požadavky aplikace

Aplikace má umožnit načtení cesty ze souboru a tu pak zobrazit v mapě. Po jejím zobrazení se vykreslí i vozidlo a to se bude podle požadavků uživatele po cestě pohybovat, při pohybu bude zobrazovat informace o své poloze.

6.2 Výběr vhodných technologií

Protože cesty jsou ukládány v GPX souboru, který je vlastně ve formátu XML, existuje několik možností jeho načtení viz kapitola 2.6. Pro načítání XML byl vybrán parser JAXB. Důvodem, proč byl vybrán, je jeho jednoduchá implementace a použití, protože se používá stejně jako obyčejná Java knihovna. K XML elementům se přistupuje pomocí tříd a metod.

Na zobrazování map v Java aplikacích existuje několik nástrojů viz kapitola 5.2. Důvodů, proč byla vybrána je hned několik:

- je open-source a protože je pod licencí LGPL je i zdarma a prakticky libovolně použitelná
- velmi snadná implementace do kódu s využitím Java knihovny Swing (viz. dále)
- oproti JMapViewer a JXMapViewer má mnohem více funkcí a mnohem méně chyb
- díky stále funkčnímu týmu vývojářů, je knihovna aktualizovaná
- knihovna si bitmapové dlaždice stahuje z OpenStreetMap serverů, takže není nutné mít obsáhlé mapy uložené na disku

6.3 Zpracování vstupu

Protože cesta je v GPX uložena jako řada zeměpisných souřadnic, je třeba je uložit do nějaké lépe pochopitelné a snadněji přístupné podoby. Při načítání jednotlivých souřadnic budou vytvářeny krátké cesty neboli úseky, tvořené vždy dvěma sousedícími

body, vznikne tak seznam úseků, kde každý má svojí délku a svůj azimut (směr od počátečního bodu ke koncovému).

Nad těmito úseky budou vytvořeny ještě virtuální cesty, jejichž počátek a konec bude určovat vzdálenost od počátku celkové cesty. Tyto cesty bude moct tvořit sám uživatel a budou sloužit rozdělení cesty na libovolné úseky nezávislé na skutečných úsecích, které bude možné používat k určování různých rychlostí na celkové cestě.

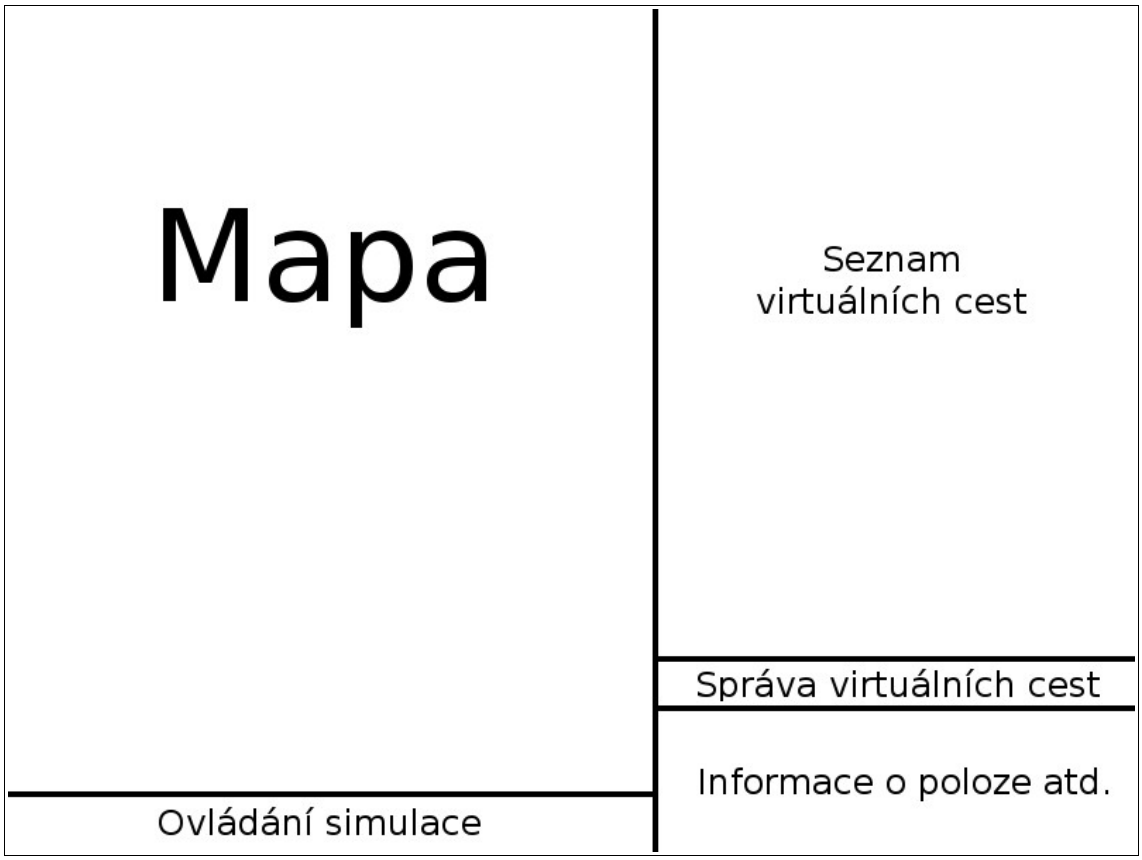
6.4 Simulace pohybu

Je možné vybírat z dvou druhů simulace viz kapitola 3. Pro tuto aplikaci byla zvolena diskrétní simulace s konstantním krokem, protože je dostačující. Délku kroku si bude uživatel moct v průběhu simulace měnit. Samotná simulace bude v nekonečném cyklu, který se bude, v případě běhu simulace, po spočtení souřadnic, pozastavovat na dobu kroku. V případě, že bude simulace pozastavena, bude simulace vždy čekat jednu sekundu, aby nebyl příliš zatěžován procesor cyklením, a pak se podívá, zda již simulace nemá běžet.

6.5 Návrh uživatelského rozhraní

Hlavní okno bude rozdělené na pět částí (obrázek 6.1):

- první a největší část bude obsahovat mapu, na které se bude vykreslovat cesta, na té se budou vykreslovat vybrané virtuální cesty a na nich bude vykresleno vozidlo
- oblast ovládání simulace bude pod mapou a budou zde tlačítka ovládající běh simulace
- další velké oblast bude obsahovat seznam virtuálních cest, zde bude možné vybírat virtuální cesty pro editaci nebo zobrazení
- správa virtuálních cest bude malá oblast pod seznamem a bude sloužit k úpravě a virtuálních cest
- poslední oblast bude pouze informační panel, ve kterém se budou zobrazovat důležité hodnoty



Obrázek 6.1 – Schéma uživatelského rozhraní

Realizační část

7. JXMapViewer2

7.1 Použití knihovny

Pro použití knihovny je nejdříve potřeba vytvořit instanci třídy reprezentující podklad komponenty, který vykresluje mapy. Poté je možné vytvořit objekt reprezentující přímo komponentu, kterou je možné vložit do okna.

```
// Vytvoří TileFactoryInfo pro OpenStreetMap
TileFactoryInfo info = new OSMTileFactoryInfo();
DefaultTileFactory tileFactory = new DefaultTileFactory(info);

// Vytvoří komponentu a přiřadí jí podklad
JXMapViewer mapa = new JXMapViewer();
mapa.setTileFactory(tileFactory);
```

Podkladu komponenty je nutné ještě nastavit počet vláken, která budou načítat dlaždice s mapou.

```
// Bude se používat 8 vláken
tileFactory.setThreadPoolSize(8);
```

Aby mapa věděla co má při spuštění zobrazit, je potřeba ještě nastavit stupeň přiblížen (čím menší číslo ,tím větší přiblížení) a místo na které se mapa vycentruje.

```
// Stupeň přiblížení se nastaví na 2 a mapa se vycentruje na budovu ZCU FAV
mapa.setZoom(2);
mapa.setAddressLocation(new GeoPosition(49.724451445, 13.350963593));
```

Nakonec už jen stačí přidat mapu jako jakoukoliv jinou komponentu do nějaké Swing komponenty.

```
// Zobrazí mapu přes celé okno
JFrame okno = new Jframe("Mapa");

okno.getContentPane().add(mapViewer);
```

Tímto se vytvoří jednoduché okno, problém je, že takto se nadá s mapou pohybovat ani ji přibližovat nebo oddalovat, proto se musí vytvořit *MouseListener* a do mapy přidat jeho jednotlivé části na naslouchání pohybu myši nebo na rotaci kolečka u myši.

```
// Vytvoření MouseListeneru
MouseInputListener mia = new PanMouseInputListener(mapa);
mapa.addMouseListener(mia);
mapa.addMouseMotionListener(mia);
mapa.addMouseListener(new CenterMapListener(mapa));
mapa.addMouseWheelListener(new ZoomMouseWheelListenerCenter(mapa));
mapa.addKeyListener(new PanKeyListener(mapa));
```

7.2 Třída GeoPosition

Tato třída reprezentuje pozici na mapě světa pomocí zeměpisné šířky a zeměpisné délky a k tomu je také v programu používána.

Pro její použití je potřeba vytvořit instanci její třídy pomocí konstruktoru. Existují dva způsoby volání konstruktoru a oba jsou si velmi podobné, proto je zde uveden pouze jeden.

```
GeoPosition pozice = new GeoPosition(49.724451445, 13.350963593);
```

Takto se vytvoří objekt se zadanými hodnotami a pro jejich získání se používají getry, které vracejí požadované hodnoty typu *double*.

```
// Vrátí zeměpisnou šířku
pozice.getLatitude();

// Vrátí zeměpisnou délku
pozice.getLongitude();
```

8. JAXB

Používá pro načítání GPX souboru, který je ve formátu XML. GPX formát má svůj XSD soubor popisující jeho elementy a atributy.

8.1 Builder

Builder pracuje tak, že ze zadaného XSD souboru, který popisuje XML soubor, vygeneruje knihovnu, k tomu využívá *ant*. Knihovna obsahuje tolik tříd kolik je elementu a každá třída má atributy reprezentující atributy elementu. Toto je jednorázová akce a není nutné ji provádět při každé kompilaci programu.

Builder je možné si napsat nebo najít na internetu, v tomto případě je builder z předmětu KIV/JXT. [3]

Spouští se příkazem v příkazové řádce:

```
> ant generovani
```

Není nutné zadávat název XSD souboru, protože builder si ho najde a zpracuje sám.

9. Vlastní realizace

V této kapitole jsou popsány všechny důležité třídy a jejich nejdůležitější metody.

9.1 Třída GPSSimulation

Toto je hlavní třída obsahující konstruktor a metody nezbytné k načítání potřebných souborů. Dále také obsahuje metody API sloužící k ovládání a získávání dat z knihovny. Seznam API metod je uveden v uživatelské příručce v příloze [číslo].

Třída obsahuje jednu statickou konstantu **JAR** typu String, která obsahuje název knihovny, sloužící k načítání XML pomocí JAXB.

9.1.1 Konstruktor

Je schopen vytvořit instanci celé knihovny, aby s ní bylo možné pracovat jako s objektem. Na základě vstupních parametrů se určí chování celé knihovny.

Parametr **zobrazitOkno** je typu boolean. Pokud je tento parametr nastaven na *true,* knihovna bude vykreslovat okno, ve kterém se zobrazí mapa a ve kterém lze knihovnu také ovládat.

Parametr **cesta** je typu String a slouží k určení cesty k souboru GPX, obsahujícího cestu, na které bude simulace probíhat.

9.1.2 Metoda nactiGPX

Tato metoda slouží k načítání GPX souboru, obsahujícího cestu, po které se v průběhu simulace bude pohybovat vozidlo. Soubor GPX je ve formátu XML a pro jeho načítání byla zvolena metoda JAXB.

Nejdříve se vytvoří instance knihovny JAXB, která obsahuje třídy a metody pro čtení XML dokumentu. Poté se vytvoří objekt třídy *Unmarshaller*, který převádí XML soubor do stromu Java objektů. Následuje vytvoření netypového kořenového elementu ze získaného stromu objektů. Pro získání typovaného kořenového elementu se použije metoda *getValue()*. Tento kořenový element obsahuje seznam elementů obsahující data, která jsou nezbytná k fungování knihovny. Seznam se získá pomocí metody *getRte()*.

```
JAXBContext jc = JAXBContext.newInstance(JAR);
Unmarshaller u = jc.createUnmarshaller();
JAXBElement<?> element = (JAXBElement<?>)u.unmarshal(new File(cesta));

GpxType koren = (GpxType)element.getValue();
List<RteType> listUseku = koren.getRte();
```

Z tohoto získaného seznamu objektů reprezentujících jednotlivé úseky cesty se následně vyextrahují pouze potřebná data a ty se uloží do seznamu objektů typu *Cesta*. První cyklus prochází jednotlivé úseky cesty a protože je každý úsek složen z posloupnosti několika bodů, je potřeba použít další cyklus k jejich procházení. Jeden objekt typu *Cesta* potřebuje ke svému určení dva body, proto se načítají vždy dva body za sebou a druhý bod je vždy použit jako první bod následující cesty.

```
List<Cesta> cesty = new ArrayList<Cesta>();
for(RteType usek : listUseku){
	List<WptType> cesticky = usek.getRtept();
	for(int i = 0; i < cesticky.size() - 1; i++){
		cesty.add(new Cesta(
			new GeoPosition(cesticky.get(i).getLat().doubleValue(),
					cesticky.get(i).getLon().doubleValue()),
			new GeoPosition(
				cesticky.get(i + 1).getLat().doubleValue(),
				cesticky.get(i + 1).getLon().doubleValue())));
	}
}
```

Takto vytvořený seznam potom metoda vrací. V případě, že metoda selže, například v důsledku neplatného vstupního souboru, vrací *null*.

9.1.3 Metoda nactiCestu

Metoda zavolá metodu *nactiGPX()*, k získání seznamu cest. Pokud se načítání povede metoda vytvoří instance dalších tříd potřebných pro simulaci, jako je seznam virtuálních cest, vozidlo, mapu a samotnou simulaci.

Metoda také detekuje, zda již nějaká simulace není spuštěna a pokud ano, je předchozí simulace vypnuta pomocí metody *interrupt()*.

9.2 Třída Cesta

Třída reprezentuje jednu krátkou úsečku reprezentovanou dvěma body. Obsahuje konstruktor, getry, setry a metody k zjišťování různých hodnot vztahujících se k úsečce. V tomto případě je úsečkou myšlena nejkratší cesta mezi dvěma body na povrchu koule.

Třída obsahuje konstantu **POLOMER**, ve které je uložen průměrný poloměr Země. Přestože „průměrný poloměr“ se může jevit jako velký zdroj chyb, není tomu tak. Úsečky jsou zpravidla tak malé, že průměrný poloměr je k získávání hodnot dostačující.

Dále třída obsahuje atributy:

- atribut **pocatek** je typu *GeoPosition* a představuje počáteční bod úsečky
- atribut **konec** je také typu *GeoPosition* a představuje koncový bod úsečky
- atribut **delka** je typu *double* a reprezentuje vzdálenost mezi počátečním a koncovým bodem v kilometrech
- atribut **smer** je typu *double* a určuje směr neboli azimut od počátečního bodu ke koncovému ve stupních

9.2.1 Konstruktor

Konstruktor má dva parametry **pocatek** a **konec**, ty ukládá do atributů a s pomocí metod *getSmer(GeoPosition)* a *getDelka(GeoPosition)* doplňuje atributy **smer** a **delka**.

9.2.2 Metoda getDelka

Toto je přetížená metoda a existuje ve dvou různých podobách:

GetDelka()

V tomto tvaru vrací pouze hodnotu atributu **delka**.

getDelka(GeoPosition)

V tomto tvaru počítá vzdálenost od bodu zadaného parametrem do koncového bodu cesty podle vzorce 4.3. Tato metoda je volána konstruktorem pro získání hodnoty pro atribut **delka**, přičemž do vstupního parametru je vložen počáteční bod.

Nejprve se spočítají rozdíly zeměpisných šířek a délek a převedou se na radiány. Poté se

převedou i zeměpisné šířky počátečního a koncového bodu. Následuje počítaní samotných rovnic a vrácení výsledné hodnoty.

```
double deltaLat = Math.toRadians(konec.getLatitude() -
pozice.getLatitude());
double deltaLon = Math.toRadians(konec.getLongitude() -
pozice.getLongitude());

double lat1 = Math.toRadians(pozice.getLatitude());
double lat2 = Math.toRadians(konec.getLatitude());

double a = Math.sin(deltaLat / 2) * Math.sin(deltaLat / 2) +
Math.sin(deltaLon / 2) * Math.sin(deltaLon / 2) * Math.cos(lat1) *
Math.cos(lat2);
double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

return POLOMER * c;
```

9.2.3 Metoda getSmer

Je to také přetížená metoda a vyskytuje se také ve dvou různých podobách:

getSmer()

Takto zadaná metoda vrací pouze hodnotu atributu **smer**.

getSmer(GeoPosition)

V tomto tvaru metoda vrací směr od bodu zadaného ve vstupním parametru ke koncovému bodu. K tomuto výpočtu používá rovnici ze vzorce 4.2. Tato metoda je volána konstruktorem pro získání hodnoty pro atribut **smer**, přičemž jako vstupní bod je zadána poloha počátečního bodu.

Nejprve se převedou zeměpisné šířky počátečního a koncového bodu ze stupňů na radiány. Poté se spočte rozdíl zeměpisných délek počátečního a koncového bodu a ten se také převede na radiány. Následně se spočtou jednotlivé parametry pro funkci *atan2()*, do které se nakonec dosadí a její výsledek se použije jako návratová hodnota.

```
double lat1 = Math.toRadians(pozice.getLatitude());
double lat2 = Math.toRadians(konec.getLatitude());
double deltaLon = Math.toRadians(konec.getLongitude() -
pozice.getLongitude());

double y = Math.sin(deltaLon) * Math.cos(lat2);
double x = Math.cos(lat1) * Math.sin(lat2) - Math.sin(lat1) * Math.cos(lat2)
* Math.cos(deltaLon);

return Math.atan2(y, x);
```

9.2.4 Metoda getPozice

Úkolem této metody je zjistit polohu bodu ležícího na úsečce v zadané vzdálenosti od počátečního bodu. Pro jeho určení se používá rovnice vzorce 4.1.

Nejprve se metoda ujistí zda je zadaná vzdálenost menší než délka úsečky tj., že leží na úsečce. Následně se převede zeměpisná šířka a délka počátečního bodu na radiány. Po získání těchto hodnost můžeme použít rovnice ze vzorce 4.1 pro výpočet hledané zeměpisné šířky a délky. Z výsledných hodnot se vytvoří bod, který je použit jako návratová hodnota.

```
if(kilometr > delka)
      return konec;
double lat1 = Math.toRadians(pocatek.getLatitude());
double lon1 = Math.toRadians(pocatek.getLongitude());

double lat = Math.asin( Math.sin(lat1)*Math.cos(kilometr/POLOMER) +
      Math.cos(lat1)*Math.sin(kilometr/POLOMER)*Math.cos(smer) );
double lon = lon1 +
      Math.atan2(Math.sin(smer)*Math.sin(kilometr/POLOMER)*Math.cos(lat1),
      Math.cos(kilometr/POLOMER)-Math.sin(lat1)*Math.sin(lat));

return new GeoPosition(Math.toDegrees(lat), Math.toDegrees(lon));
```

9.3 Třída VirtuálniCesta

Tato třída reprezentuje libovolně dlouhou cestu, je složená z několika úseček typu *Cesta* nebo jejich částí. Její počátek a konec není dán jednotlivými body, nýbrž vzdálenostmi od úplného počátku. Je používána pro vytváření úseků s různými rychlostmi, jakými se po nich vozidlo bude pohybovat. Třída obsahuje konstruktor, getry, setry a metodu umožňující řazení virtuálních cest.

Ve třídě je jedna konstanta **DEF_RYCHLOST**, která představuje výchozí rychlost, kterou se po virtuální cestě jezdí.

Atributy třídy:

- atribut **pocatek** je typu *double* a značí v kolikátém kilometru celkové délky cesty daná virtuální cesta začíná
- atribut **konec** je typu *double* a značí v kolikátém kilometru celkové délky cesty daná virtuální cesta končí
- atribut **rychlost** je také typu *double* a určuje rychlost, kterou se po dané cestě jezdí a je v kilometrech za hodinu
- atribut **kresleni** je typu *KresliCestu* a slouží k vykreslování virtuální cesty do mapy

9.3.1 Konstruktor

U této třídy je konstruktor přetížený a lze ho volat ve dvou různých formách:

VirtualniCesta(double pocatek, double konec, List<Cesta> cesty)

Tento konstruktor pouze volá druhý typ konstruktoru, kde do jeho čtvrtého parametru **rychlost** vloží hodnotu **DEF_RYCHLOST**.

VirtualniCesta(double pocatek, double konec, List<Cesta> cesty, double rychlost)

První dva parametry konstruktoru **pocatek** a **konec** jsou hodnoty, které se uloží do atributů **pocatek** a **konec**. Třetí parametr **cesty** představuje seznam všech cest, ze kterých je tvořena celková cesta. Poslední parametr je **rychlost**, kterou se po dané virtuální cestě jezdí a která se uloží do atributu **rychlost**.

Po uložení hodnot do atributů se vytvoří seznam nových cest do kterého se budou vkládat úseky, kterými je virtuální cesta tvořena. Dále se vytvoří proměnné, které pomohou seznam naplnit. Proměnná **ujeto** reprezentující vzdálenost, která již byla zkontrolována, proměnná **index**, která slouží k procházení seznamu cest, proměnné **maPocatek** a **maKonec** představují, zda již byl nalezen počátek nebo konec.

Následuje cyklus procházející seznam úseků **cesty** a bude trvat dokud nebude nalezen konec. Na začátku každého cyklu se k proměnné **ujeto** přičte délka právě procházené cesty a na konci se vždy inkrementuje **index**. Uvnitř cyklu jsou pak tři podmínky rozhodující o tom, zda se přidá jen koncová část cesty nebo počáteční část cesty nebo se cesta přidá celá, jinak se nestane nic, tedy žádná cesta se nepřidá. Pro získávání částí cest se používá metoda *getPozice()* ze třídy *Cesta*. Pokud je nalezena a vložena počáteční cesta je proměnná **maPocatek** nastavena na *true* a pokud je nalezena koncová cesta je proměnná **maKonec** nastavena na *true* a cyklus se ukončí.

Nakonec se, s pomocí nového seznamu cest, vytvoří objekt typu *KresliCestu* a uloží se do atributu **kresleni**.

9.4 Třída KresliCestu

Tato třída představuje jednoduchý painter, který je schopný na mapu kreslit jakékoliv cesty se mu zadají. Třída obsahuje pouze konstruktor a override metodu, která vznikne implementací Java rozhraní Painter.

Privátní atributy třídy jsou:

- atribut **cesty**, který je typu *List<Cesta>*, tento seznam obsahuje seznam cest, které se budou na mapě vykreslovat
- atribut **barva** je typu *Color* a značí barvu, kterou bude cesta vykreslena
- atribut **tloustka** je typu *int* a určuje tloušťku čáry reprezentující cestu

9.4.1 Konstruktor

Konstruktor pouze ukládá parametry do atributů. Parametr **cesty** do atributu **cesty**, parametr **barva** do atributu **barva**, Parametr **tloustka** do atributu **tloustka**.

9.4.2 Metoda paint

Metoda má sice čtyři parametry, ale ty se zadávají sami při kreslení. Jinak metoda dokáže vykreslovat na mapu knihovny JXMapViewer2 cesty načtené z GPX nebo virtuální cesty.

V první řadě je potřeba zjistit velikost kreslícího plátna neboli velikost mapy, ta může být pokaždé jiná, záleží totiž na stupni přiblížení. Čím je stupeň větší, tím je mapa vzdálenější a kreslící plátno se zmenšuje. Po zjištění velikosti se kreslící plátno posune do počátku soustavy souřadnic. Následuje nastavení barvy a tloušťky čáry, která se bude kreslit. Poté se v cyklu začnou procházet všechny cesty. U každé cesty se nejdříve zjistí, kde na plátně se nachází počáteční a koncový bod, k tomu pomáhá metoda z knihovny JXMapViewer2 *geoToPixel()*, která dokáže převést zeměpisné souřadnice na pozici na plátně v pixelech. Nakonec se získaným počátečním a koncovým bodem se nakreslí čára.

```
Rectangle obdelnik = object.getViewportBounds();
g.translate(-obdelnik.x, -obdelnik.y);

g.setColor(barva);
g.setStroke(new BasicStroke(tloustka));
for(Cesta c : cesty){
      Point2D bod = object.getTileFactory().geoToPixel(c.getPocatek(),
                        object.getZoom());
      Point2D bod2 = object.getTileFactory().geoToPixel(c.getKonec(),
                        object.getZoom());

      g.drawLine((int)bod.getX(), (int)bod.getY(), (int)bod2.getX(),
                        (int)bod2.getY());
}
```

9.5 Třída KresliZoomObdelnik

Je to třída fungující na stejném principu jako třída *KresliCestu*. Na mapu světa nakreslí obdélník určující část mapy, která se zobrazí při přiblížení kolečkem myši. Tento painter lze libovolně zapínat a vypínat.

9.6 Třída ListVirtuálníchCest

Tato třída reprezentuje seznam virtuálních cest a umožňuje s ním pracovat a různě ho upravovat. Obsahuje konstruktor, getry a setry, metody typické pro práci se seznamem a metody umožňující seznam snadno upravovat, tyto metody pracují tak aby v seznamu nevznikali mezery tedy, aby celý seznam dohromady pokrýval celou cestu.

Atributy třídy:

- atribut **virtualniCesty** je seznam všech virtuálních cest a je typu *List<VirtualniCesta>*
- atribut **listTextu** je seznam textových řetězců popisujících virtuální cesty a je určen k zobrazování v JListu v okně a je typu *DefaultListModel<String>*
- atribut **cesty** je typu *List<Cesta>* a představuje seznam všech cest

9.6.1 Konstruktor

Vytvoří prázdný seznam virtuálních cest a prázdný seznam textů. A jeho jediný parametr **cesty** uloží do atributu **cesty.**

9.6.2 Metody pridej a odeber

Metody pracují stejně jako by byli u normálního seznamu s tím rozdílem, že při přidávání a odebírání položek pracují se dvěma seznamy, podle očekávání se seznamem virtuálních cest a navíc se seznamem textových řetězců.

9.6.3 Metoda rozdelCestu

Metoda rozděluje zadanou virtuální cestu, která je v seznamu virtuálních cest, podle zadané vzdálenosti od počátku dané cesty.

Nejprve se do pomocných proměnný uloží atributy zadané cesty a její index v seznamu. Poté se zadaná cesta ze seznamu smaže a je nahrazena dvěma novými cestami, které vzniklou mezeru vyplní. Nakonec se volá metoda pro řazení, protože dvě nové cesty se přidají na konec seznamu a seznam by tedy nebyl seřazený.

```
double pocatek = vc.getPocatek();
double konec = vc.getKonec();
double rychlost = vc.getRychlost();
int index = virtualniCesty.indexOf(vc);

virtualniCesty.odeber(index);
listTextu.odeber(index);

pridej(new VirtualniCesta(pocatek, pocatek + kilometr, cesty, rychlost));
pridej(new VirtualniCesta(pocatek + kilometr, konec, cesty, rychlost));
serad();
```

9.6.4 Metoda spojCesty

Jediným parametrem této metody je seznam řetězců reprezentující seznam virtuální cesty. Podmínkou je, aby virtuální cesty byly seřazeny a po sobě jdoucí tzn., aby mezi cestami nebyli mezery a mohla tak vzniknout jedna spojitá cesta.

Nejdříve se zjistí počáteční vzdálenost tak, že se ze seznamu vybere první prvek a jeho počáteční hodnota. Poté se získá konečná vzdálenost tak, že se vezme koncová vzdálenost posledního prvku. Poté se do seznamu vytvoří nová cesta se získaným počátkem a koncem a nepotřebné cesty se ze seznamu vymažou. Nakonec se seznam ještě musí seřadit.

```
double pocatek =

virtualniCesty.get(listTextu.indexOf(seznamCest.get(0))).getPocatek();
double konec = virtualniCesty.get(listTextu.indexOf(seznamCest.get(
      seznamCest.size() - 1))).getKonec();
pridej(new VirtualniCesta(pocatek, konec, cesty));
```

```
for(int i = 0; i < seznamCest.size(); i++){
	odeber(seznamCest.get(i));
}
sort();
```

9.6.5 Metoda zmenRychlost

Jejím prvním parametrem je seznam řetězců reprezentující seznam virtuálních cest. Tento seznam může obsahovat libovolné virtuální cesty ze seznamu v jakémkoliv pořadí. Metoda všem těmto cestám přiřadí novou rychlost zadanou druhým parametrem.

Při procházení jednotlivých řetězců se určí k nim patřící virtuální cesta a té se přenastaví její rychlost. Na konci je nutné seznamy virtuálních cest a řetězců synchronizovat, protože se změnili atributy cest a řetězce tím pádem již nejsou platné.

```
for(String s :seznamCest){
	virtualniCesty.get(listTextu.indexOf(s)).setRychlost(rychlost);
}
synchronizuj();
```

9.6.6 Metoda synchronizuj

V případě že dojde k porušení integrity seznamů, je nutné zavolat tuto metodu která podle listu virtuálních cest znovu vytvoří seznam řetězců, které je reprezentují. K porušení integrity dochází, když se v seznamu s cestami něco změní, například rychlost.

```
listTextu.removeAllElements();
for(VirtualniCesta vc : virtualniCesty){
	listTextu.addElement(vc.toString());
}
```

9.7 Třída Vozidlo

Tato třída reprezentuje nějaký pohyblivý objekt z reálného světa, například automobil, kterým se bude pohybovat po mapě. Tato třída implementuje painter, slouží tedy zároveň jako nástroj na zakreslení vozidla do mapy. Třída obsahuje konstruktor, který

pouze ukládá parametry do atributů, getry, setry a metodu *paint()*, která vozidlo vykresluje na mapu.

Jediná konstanta třídy se názvem **VELIKOST** představuje velikost kresleného objektu v pixelech.

Atributy třídy:

- atribut **pozice** je typu *GeoPosition* a představuje polohu vozidla na mapě
- atribut **smer** je typu *double* a představuje směr neboli azimut, jakým se vozidlo pohybuje

9.7.1 Metoda paint

Tato metoda je override a její parametry není nutné nikdy zadávat, protože metoda se volá automaticky při vykreslování.

Stejně jako u předchozích painterů se nejdříve zjistí velikost plátna a poté se posune do počátku soustavy souřadnic. Nastaví se barva a tloušťka čáry, kterou se bude kreslit a následně se zjistí pomocí metody *geoToPixel()*, kde se nachází vozidlo podle jeho polohy. Následně se plátno posune tak, aby střed soustavy souřadnic byl v poloze vozidla a to z toho důvodu, aby bylo možné jednoduše provést rotaci určující směr pohybu vozidla. Poté už se jen nakreslí kružnice a šipka a vozidlo je hotové.

```
Rectangle obdelnik = object.getViewportBounds();
g.translate(-obdelnik.x, -obdelnik.y);

g.setColor(Color.BLACK);
g.setStroke(new BasicStroke(4));

Point2D bod = object.getTileFactory().geoToPixel(pozice, object.getZoom());
g.translate((int)bod.getX(), (int)bod.getY());
g.rotate(smer);

g.drawOval(-(VELIKOST / 2), -(VELIKOST / 2), VELIKOST, VELIKOST);
g.drawLine(0, 0, 0, -50);
g.drawLine(0, -50, -10, -40);
g.drawLine(0, -50, 10, -40);
```

9.8 Třída Mapa

Tato třída slouží pro práci s mapou poskytnutou knihovnou JXMapViewer2. Obsahuje konstruktor, jeden getr a metody pro práci s paintery. Metody pro práci s paintery jsou tu proto, že zachovávají požadované pořadí painterů ve frontě, protože na pořadí záleží. Painter první pořadí se vykresli jako první a ti, kteří se budou vykreslovat po něm ho překreslí, což je u některých nežádoucí, například vozidlo by mělo vždy být kresleno nad cestou a ne pod ní.

Atributy třídy:

- atribut **mapa** je typu *JXMapViewer* a je to komponenta, která vykresluje mapu a kterou je možné vložit do okna
- atribut **okno** je typu *Okno* a je to ukazatel na hlavní okno a slouží především k získávání důležitých dat

9.8.1 Konstruktor

Kromě toho, že konstruktor přiřazuje parametr **okno** do atributu **okno**, tak navíc vytváří a nastavuje komponentu pro vykreslování mapy. Toto vytváření a nastavování probíhá viz kapitola 6.1 a následně se zavolá metoda *kresli()*.

9.8.2 Metoda kresli

Tato metoda slouží k vykreslování méně důležitých painterů, které se budou zobrazovat v nejnižších vrstvách a které se už v průběhu programu nemění.

Nejprve se vytvoří instance tříd, které mají být přidány do fronty. Poté se vytvoří samotná fronta, do které se paintery přidají. Následně se všechny sloučí do jednoho a ten se předá komponentě vykreslující mapu, která si je už sama bude vykreslovat.

```
KresliZoomObdelnik obdelnik = new KresliZoomObdelnik(okno);
List<Painter<JXMapViewer>> painters = new ArrayList<Painter<JXMapViewer>>();
      painters.add(obdelnik);
CompoundPainter<JXMapViewer> painter = new
      CompoundPainter<JXMapViewer>(painters);
mapa.setOverlayPainter(painter);
```

9.8.3 Metoda kresliCestu

Tato metoda se volá až po tom, co je nahrána cesta ze souboru GPX a přidává do fronty paintery vykreslující cestu a vozidlo.

9.8.4 Metoda pridejPainter

Tato metoda umožňuje do fronty přidávat další paintery a při tom dodržet, aby painter vozidla zůstal jako poslední, aby nebylo vozidlo nikdy vykresleno pod cestami. Používá se pro kreslení virtuálních cest, protože ty se můžou v průběhu simulace měnit, přidávat a odebírat.

Nejdříve se vytvoří iterátor umožňující procházení všech painterů, potom se pomocí cyklu a metody *hasNext()* iterátor přesune na poslední pozici ve frontě. Tam se nachází painter vozidla, ten se uloží do proměnné a smaže se z fronty. Poté se už jen přidá požadovaný painter a následně painter vozidla.

```
Iterator<Painter<JXMapViewer>> iterator =
((CompoundPainter<JXMapViewer>)mapa.getOverlayPainter()).getPainters().
      iterator();

Painter<JXMapViewer> entyta = null;
while(iterator.hasNext()) entyta = iterator.next();
Painter<JXMapViewer> vozidlo = entyta;

((CompoundPainter<JXMapViewer>)mapa.getOverlayPainter()).
      removePainter(vozidlo);
((CompoundPainter<JXMapViewer>)mapa.getOverlayPainter()).
      addPainter(painter);
((CompoundPainter<JXMapViewer>)mapa.getOverlayPainter()).
      addPainter(vozidlo);
```

9.9 Třída Simulace

V této třídě probíhá samotná simulace pohybu vozidla po cestě zadané GPX souborem. Třída dědí od třídy Thread, aby mohla simulace běžet v samostatném vláknu. Je zde konstruktor, getry, setry, metoda *run()* a metoda pro změnu polohy vozidla.

Dále třída obsahuje několik konstant:

- konstanta **PLAY** je typu *integer* a značí stav simulace, ve kterém je simulace spuštěna a vozidlo se pohybuje po mapě
- konstanta **PAUZA** je také typu *integer* a značí stav, ve kterém je simulace pozastavena a vozidlo stojí na místě
- konstanta **STOP** je typu *integer* a značí stav, ve kterém je simulace zastavena a vozidlo se vrátí na začátek cesty

Atributy třídy:

- atribut **cesty** je seznam všech úseků tvořící celkovou cestu a je typu *List<Cesta>*
- atribut **vitualniCesty** je seznam virtuálních cest a je typu *ListVirtualnichCest*
- atribut **vozidlo** je typu *Vozidlo* a představuje objekt pohybující se po mapě
- atribut **celkovaDelka** je typu *double* a je to celková délka cesty, po které se vozidlo pohybuje v kilometrech
- atribut **ujetaVzdalenost** je typu *double* a značí, kolik kilometrů již vozidlo po cestě ujelo v kilometrech
- atribut **mapa** je typu *Mapa* představuje mapu
- atribut **okno** je typu *Okno* a představuje hlavní okno programu

9.9.1 Konstruktor

Konstruktor ukládá všechny své parametry do atributů a počítá celkovou délku cesty tak, že sečte délky všech úseků. Následně do seznamu virtuálních cest přidá první virtuální cestu, která pokrývá celou cestu. Nakonec ještě pošle do okna potřebné hodnoty, aby okno zobrazovalo korektní hodnoty.

9.9.2 Metoda run

Metoda obsahuje nekonečnou smyčku, aby vlákno běželo tak dlouho, dokud bude program spuštěn. V této smyčce se se rozhoduje o tom co se bude dít. Existují totiž tři

stavy, ve kterých se simulace může nacházet: stav běžící, pozastavený a zastavený. O tom v jakém stavu se nachází se dozvídá z objektu **okno**, ve kterém se dá simulace řídit.

Stav běžící znamená, že simulace probíhá a vozidlo se pohybuje po mapě nenulovou rychlostí, pokud mu tedy nebyla nastavena rychlost nula. V průběhu simulace se mění ujetá vzdálenost podle rychlosti a doby od posledního přepočítání. Po spočtení nové ujeté vzdálenosti se zavolá metoda *pohniSe()*.

Ve stavu pozastavený vlákno nedělá nic a ve stavu zastavený se vozidlo vrátí na počátek cesty a simulace se změní do stavu pozastavený.

9.9.3 Metoda pohniSe()

Na základě ujeté vzdálenosti počítá novou polohu vozidla.

Metoda nejdříve v cyklu vyhledá na jakém úseku cesty se vozidlo podle ujeté vzdálenosti má nacházet. To probíhá tak, že se k lokální proměnné **ujeto** přičítají vzdálenosti procházených cest, dokud jeho hodnota není větší než ujetá vzdálenost. Zároveň se při tom do proměnné **i** zaznamenávají indexy procházených cest. Následně se kontroluje zda je ujetá vzdálenost menší než celková délka cesty. Pokud ano, vozidlu se nastaví poloha na získaném úseku pomocí metody *getPozice(double)*. Pokud ne, ujetá vzdálenost se nastaví na celkovou délku cesty, poloha vozidla se nastaví na koncoví bod poslední cesty a simulace se nastaví na stav pozastaveno.

```
double ujeto = 0;
int i = 0;
while(ujeto + cesty.get(i).getDelka() < ujetaVzdalenost){
	ujeto += cesty.get(i).getDelka();
	i++;
	if(i >= cesty.size())
		break;
}
if(i < cesty.size() && ujetaVzdalenost < celkovaDelka){
	vozidlo.setPozice(cesty.get(i).getPozice(ujetaVzdalenost - ujeto),
		cesty.get(i));
}
```

```
else{
	vozidlo.setPozice(cesty.get(cesty.size() - 1).getKonec(), cesty.get(
		cesty.size() - 1));
	ujetaVzdalenost = celkovaDelka;
	okno.setStavSimulace(PAUZA);
}
```

9.10 Třída Okno

Toto je velmi komplexní třída, ale většina jejích metod je nezajímavá, protože slouží pouze k vykreslování nebo ovládání požadovaných komponent.

Nicméně má dva důležité atributy:

- atribut **rychlostObnovySimulace** je typu *integer* a značí délku kroku v milisekundách, po kterém se přepočítají všechny hodnoty v simulaci
- atribut **stavSimulace** je typu *ineger* a značí stav simulace, může nabývat hodnot daných konstantami ve třídě *Simulace*

V této třídě jsou umístěné z toho důvodu, že je lze pomocí GUI měnit.

10. Testování

10.1 Přesnost double

I když je *double* velmi přesný má své limity a je potřeba zjistit, jak velké pohyby po Zemi je možné modelovat. Jde tedy o to zjistit, jak malé číslo je možné přičíst k zeměpisným souřadnicím, aby to mělo nějaký vliv, a poté, jak velká změna je to ve vzdálenosti na Zemi.

Nejmenší číslo, které je možné přičíst v intervalu od 0 do 360 (interval pokrývající možné hodnoty zeměpisných souřadnic), je 10^{-13}. To bylo zjišťováno následujícím algoritmem, kde za hodnotu *hodnota* byla dosazována různá čísla od -16, dokud nebyl počet chyb roven nule.

```
for(int i = 0; i < 360; i++){
	double cislo = i + Math.pow(10, -13);

	//pokud je číslo i po přičtení stejné jako předchozí, nastala chyba
	if(i == cislo)
		chyba++;
}
System.out.println("Chyb:  " + chyba);
```

Pokud se vezme náhodná zeměpisná souřadnice *pocatek* a vytvoří se druhá, která bude mít zeměpisnou šířku posunutou o hodnotu získanou v předchozím algoritmu, tak vzdálenost mezi nimi je zhruba v řádech 10^{-5} milimetru. Pro zeměpisnou délku vyjde ještě menší hodnota. Z toho plyne, že teoreticky je možné simulovat pohyb i v jednotkách menších než milimetr.

```
//změření vzdálenosti mezi počátkem a nově vytvořenou souřadnicí
double presnost = getDelka(pocatek, new GeoPosition(pocatek.getLatitude() +
Math.pow(10, -13), pocatek.getLongitude())); //km

System.out.println("Double presnost:" + presnost);
```

10.2 Přesnost výpočtů

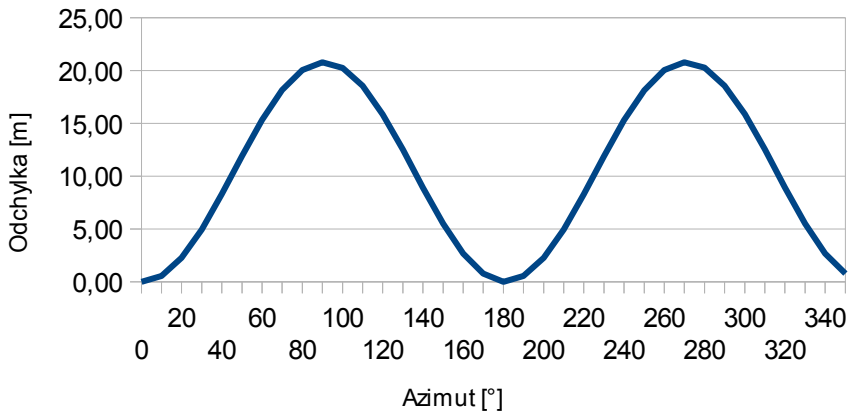
Protože se ve vzorcích vyskytují goniometrické funkce, které jsou častými tvůrci chyb a nepřesností, je nutné zjistit jakou mírou ovlivňují výsledek. Testování bude probíhat tak, že se z jedné zeměpisné souřadnice bude pohybovat do všech směrů a vracet se zpět a poté zjišťovat o jakou vzdálenost se výsledná souřadnice liší od počáteční.

```
double delka = 10; //km
for(int i = 0;  i < 360; i++){
	//pohyb od středu
	GeoPosition konec = dejBod(pocatek, i, delka);
	//pohyb zpět
	GeoPosition zpet = dejBod(konec, (i + 180) % 360, delka);
	//výpis vzdálenosti mezi body
	System.out.println(getDelka(pocatek, zpet));
}
```

Kde metoda *dejBod()* získává novou zeměpisnou souřadnici vzdálenou od počátečního bodu (určeného prvním parametrem) hodnotou v kilometrech danou třetím parametrem a s azimutem daným druhým parametrem.

```
private static GeoPosition dejBod(GeoPosition vozidko, double smer, double
delka){
	smer = Math.toRadians(smer);
	double lat1 = Math.toRadians(vozidko.getLatitude());
	double lon1 = Math.toRadians(vozidko.getLongitude());
	double lat2 =  Math.asin( Math.sin(lat1)*Math.cos(delka/POLOMER) +
		Math.cos(lat1)*Math.sin(delka/POLOMER)*Math.cos(smer) );
	double lon2 = lon1 + Math.atan2(
		Math.sin(smer)*Math.sin(delka/POLOMER)*Math.cos(lon1),
		Math.cos(delka/POLOMER)-Math.sin(lon1)*Math.sin(lat2));
	return new GeoPosition(Math.toDegrees(lat2), Math.toDegrees(lon2));
}
```

Do každého směru vznikají různě velké chyby. Při pohybu do vzdálenosti 10 kilometrů (při pohybu tam i zpět je to dohromady 20 kilometrů) vzniká chyba dosahující odchylky až 20 metrů viz graf 11.1.



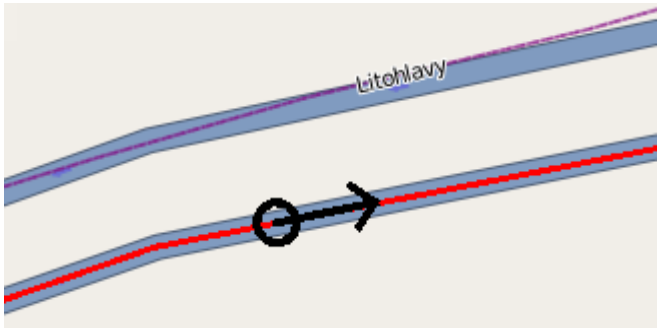
Graf 11.1 – Chyby v různých směrech

Čím jsou pohyby větší, tím je větší i chyba, nicméně průměrná délka pohybu při simulaci pohybu po cestě je asi 0.2 kilometru a tam je odchylka menší než jeden centimetr, což je přípustná chyba.

10.3 Zobrazení

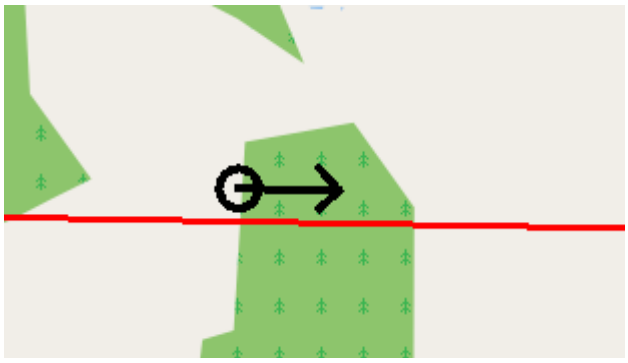
Jde o to zkontrolovat zda se v průběhu simulace auto udržuje na cestě, po které se má pohybovat a nevybočuje z ní i při maximálním přiblížení.

Výsledky testování typických cest generovaných různými programy jsou uspokojivé a simulace s chová podle očekávání viz obrázek 10.1



Obrázek 10.1 – Vozidlo na cestě

Nicméně při vytvoření atypické cesty takové, že její úseky jsou nesmyslně veliké, dochází k tomu, že vozidlo začíná mírně vyjíždět z projížděného úseku, ale vždy projede jeho začátkem a koncem, viz obrázek 10.2.



Obrázek 10.2 – Vozidlo mimo cestu

Tento jev je způsoben tím, že cesty se kreslí v kartézský souřadnicích na plátně, kdežto výpočet pohybu pracuje se souřadnicemi sférickými. Kreslící plátno vyznačuje nejkratší cestu mezi body na plátně, což je přímka, ale nejkratší vzdálenost mezi body ve skutečném světě po povrchu planety je jiná. Takže tento problém ve skutečnosti není

chyba při výpočtu nýbrž chyba v zobrazení.

Průměrná délka úseku je zhruba 0.2 kilometru a vozidlo začne mírně vyjíždět až při 5 kilometrovém úseku. Čím větší úsek a čím blíže k pólům tím větší odchylka.

11. Závěr

Cílem bakalářské práce bylo vytvoření programu schopného simulovat pohyb nějakého objektu po povrchu Země. K určování cesty, po které se bude objekt pohybovat byl zvolen obecně známí formát GPX pro ukládání cest. Je tedy možné cesty vytvářet pomocí programů, kterých je na internetu mnoho a jsou zdarma nebo použít cesty vytvořené jinými uživateli. Po načtení cesty do programu je možné cestu libovolně rozdělovat na menší úseky a těm nastavovat rozdílné rychlosti.

Simulaci je možné nastavovat různé parametry a měnit tak její chování podle potřeb uživatele. V průběhu testování byla ověřena funkčnost programu, ten se choval tak jak se od něj očekávalo. Proto byl na konec program převeden na knihovnu a bylo mu vytvořené API, aby ho bylo možné snadno používat v kombinaci s dalšími aplikacemi.

Přehled zkratek

GPS – globální polohový systém

GIS – globální informační systém

XML – obecně známí značkovací jazyk užívaný k výměně informací

GUI – grafické uživatelské rozhraní

Literatura

[1] U.S. GOVERNMENT. *GPS.gov* [online]. [cit. 2013-04-26]. Dostupné z: http://www.gps.gov/

[2] *OpenStreetMap Wiki* [online]. [cit. 2013-04-26]. Dostupné z: http://wiki.openstreetmap.org/

[3] HEROUT, Pavel. *Přednášky KIV/JXT* [online]. [cit. 2013-04-26]. Dostupné z: http://www.kiv.zcu.cz/~herout/vyuka/jxt/prednasky/jxt-1a4.pdf

[4] *Modelování a simulace* [online]. [cit. 2013-04-26]. Dostupné z: http://school.kjn.cz/modelovani-simulace/

[5] *JXMapViewer2* [online]. [cit. 2013-04-26]. Dostupné z: https://github.com/msteiger/jxmapviewer2

[6] *Historie a vznik systému GPS* [online]. [cit. 2013-05-02]. Dostupné z: http://www.beruna.cz/text-historie-a-vznik-systemu-gps/

[7] *Global Positioning Systems* [online]. [cit. 2013-05-04]. Dostupné z: http://web.archive.org/web/20110719232148/http://www.macalester.edu/~halverson/math36/GPS.pdf

[8] *Calculate distance, bearing and more between Latitude/Longitude points* [online]. [cit. 2013-05-04]. Dostupné z: http://www.movable-type.co.uk/scripts/latlong.html

[9] *Osm4J* [online]. [cit. 2013-05-04]. Dostupné z: http://osm4j.sourceforge.net/

Přílohy

Příloha 1 - Uživatelská příručka

Pro použití knihovny je důležité vytvořit instanci třídy GPSSimulation, kterou je potřeba ještě před tím naimportovat.

Konstruktor třídy má dva parametry:

showWindow – Pokud je tento parametr *true*, zobrazí se okno umožňující ovládání pomocí GUI a API. Pokud je parametr *false*, žádné okno se nezobrazí a je možné ovládání pouze pomocí API.

gpxFilePath – Cesta k souboru .gpx s cestou. Může být *null*, pokud není **showWindow** false.

GUI

Menu

Soubor → Načti cestu

Otevře dialogové okno umožňující vybrat .gpx soubor s cestou.

Soubor → Konec

Zavře okno a ukončí simulaci

Nastavení → Zobrazit obdélník přiblížení

Vykreslí v mapě obdélník označující oblast, která se zobrazí při přiblížení kolečkem.

Nastavení → Rychlost simulace

Nastavuje dobu prodlevy mezi přepočty simulace.

Pomoc → Vytvořit cestu

Otevře internetovou adresu v internetovém prohlížeči, na které je možné vytvářet cesty a stahovat je do počítače.

Pomoc → O programu

Informace o programu.

Ovládání mapy

Pohyb mapy:

Kliknutím levým tlačítkem a posouváním myši se mapa pohybuje.

Přibližování a oddalování:

Je možné provést kolečkem na myši. Přiblížení je pak možné i pomocí dvojkliku levým tlačítkem.

Správa virtuálních cest

Rozdělení cest:

Tlačítko „Rozděl cestu“ otevře dialogové okno, kde je možné nastavit, na kolikátém kilometru se cesta rozdělí.

Spojení cest

Tlačítko „Spoj cesty“ spojuje vybrané cesty do jedné.

Změna rychlosti na virtuálních cestách

Tlačítko „Nastav rychlost“ otevře dialogové okno, kde je možné nastavit novou rychlost vybraných cest.

Ovládání simulace

Tlačítko „Play“ spustí simulaci.

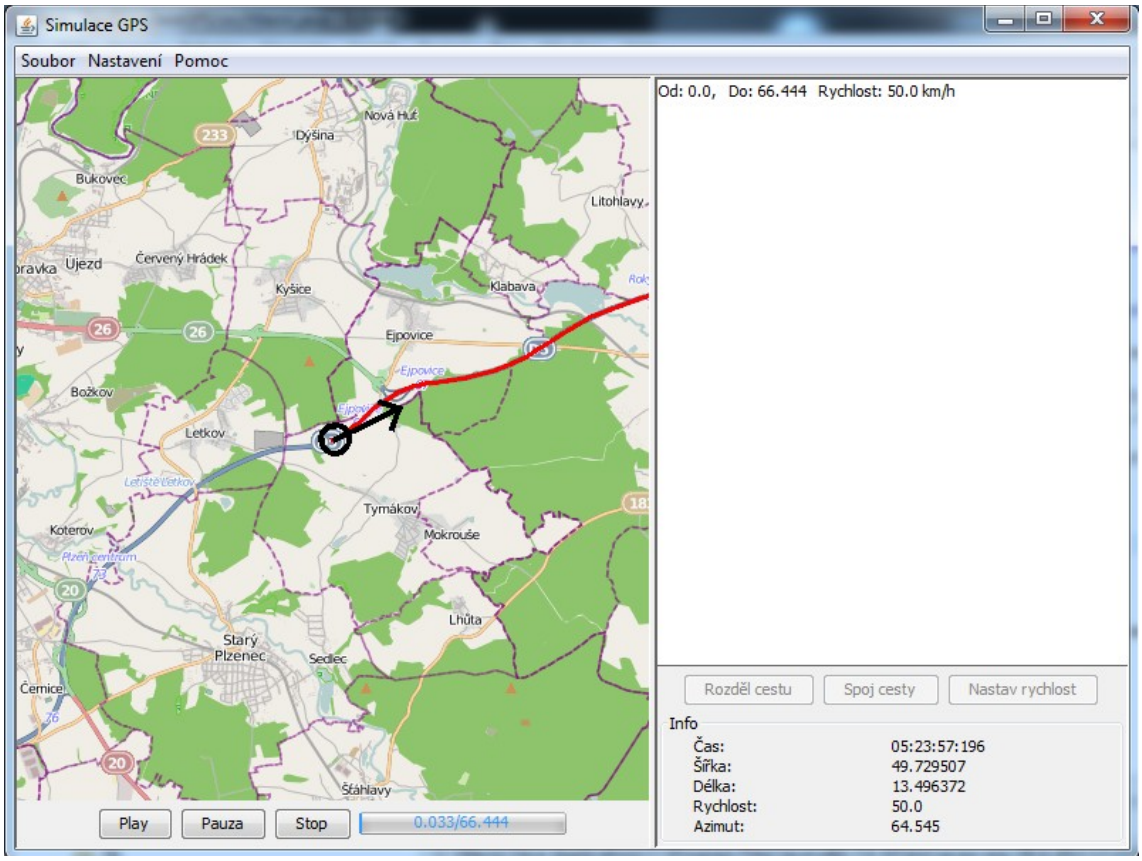
Tlačítko „Pauza“ pozastaví simulaci.

Tlačítko „Stop“ zastaví simulaci a vrátí vozidlo na počátek cesty.

API

GeoPosition	getCoordinates()	Vrátí aktuální souřadnice vozidla.
GeoPosition	getCoordinates(distance, absolute)	Posune vozidlo o zadanou vzdálenost *distance*. Pokud je *absolute* rovno ture, posouvá se od začátku, jinak od aktuální polohy. Výslednou novou polohu vrátí.
GeoPosition	getCoordinates(time, speed, absolute)	Posune vozidlo podle doby jízdy *time* a jeho rychlosti *speed*. Pokud je *absolute* rovno ture, posouvá se od začátku, jinak od aktuální polohy. Výslednou novou polohu vrátí.
double	getTrackLenght()	Vrátí celkovou délku cesty v metrech.
double	getDistanceTraveled()	Vrátí vzdálenost v metrech, kterou již vozidlo ujelo.
-	runSimulation()	Vytvoří vlákno a pustí v něm simulaci, která je však pozastavená.
-	setSimulationRunning(running)	Pokud je *running* ture, tak simulaci spustí, a pokud je false, tak simulaci pozastaví.
-	setSpeed(speed)	Nastaví rychlost *speed* všech virtuálních cest v metrech za sekundu, tedy i celkovou rychlost vozidla kdekoliv na dráze.
-	setRecaulculateDelay(millis)	Nastaví dobu *millis* v milisekundách, po které bude docházet k přepočtům simulace.
-	loadRoute(pathToFile, runSimulation)	Načte soubor daný cestou v *pathToFile*. Pokud se to povede inicializuje se vozidlo, zakreslí se načtená cesta do mapy a vycentruje se. V případě, že je *runSimulation* je *ture*, tak se spustí simulace.

Obrázek GUI



Příklad použití API

```
//Vytvoření instance třídy
GPSSimulation simulace = new GPSSimulation(true, cesta);
//Vypíše souřadnice na počátku cesty
System.out.println("Zacatek: " + simulace.getCoordinates().toString())
//Posune vozidlo o 50 metrů od aktuální polohy a tu vypíše.
System.out.println(simulace.getCoordinates(50, false).toString());
//Načtení úplně nové cesty
simulace.loadRoute(cesta2, true);
//Nastaví rychlost vozidla
simulace.setSpeed(50);
//Vytvoří vlákno se simulací
simulace.runSimulation();
//Přepne simulaci do stavu běžící
simulace.setSimulationRunning(true);

//V cyklu bude každou sekundu odečítat pozici vozidla.
while(true){
	System.out.println(simulace.getCoordinates());
	System.out.println("Ujeto: " + simulace.getDistanceTraveled() + " z "
		+ simulace.getTrackLenght());
	Thread.sleep(1000);
}
```

Obsah CD

Složka **knihovna** obsahuje výsledný produkt práce.

Složka **src** obsahuje zdrojové kódy projektu.

Složka **texty** obsahuje bakalářskou práci.